CS-doc-5406

CMS Intel Phi Investigation

Liz Sexton-Kennedy and Stephan Lammel

August 19th, 2014
version 1.02

Intel's current Xeon Phi coprocessor cards provide up to a thousand GFLOPS of double precision performance, i.e. about four times current server processor performance. Investigating the use of the technology for compute intensive high-energy-physics applications is thus natural.

The Compact Muon Solenoid experiment observes proton–proton collisions at a rate of 40 MHz, records about 2 PetaBytes of data per year, and utilizes around 100,000 CPU cores at any given time to reconstruct, simulate, and analyze this data.

A first investigation of the technology run into obstacles compiling the complex reconstruction, simulation, and analysis software of the experiment with the Intel C++ compiler. The experiment software uses extensions provided by the C++11 standard that the Intel compiler does not (yet) support. A stand-alone Monte Carlo generator, capable of utilizing many cores, showed an Intel Xeon Phi core to perform significantly slower than a current server processor core.

1 Introduction

At the end of 2012 Intel released a new Xeon Phi coprocessor [1], based on its Many Integrated Core (MIC) architecture. The 5110P coprocessor card plugs into a PCI Express slot on the motherboard. The coprocessor is effectively a 60 core SMP processor with four hardware threads per core for a total of 240 virtual or logical cores. Each core has its own 64-Byte wide vector floating point unit. The coprocessor runs at 1.053 GHz and has 32 kBytes level-1 and 512 kBytes of level-2 cache per core. There is 8 GBytes of memory on the coprocessor card. The theoretical maximum double precision performance is just over a Tera FLoating-point Operations Per Second, TFLOPS. In comparison, current Intel and AMD server processors can perform about a quarter TFLOPS.

The instruction set of the first generation Intel Phi coprocessor provides a 64-bit execution environment with 512-bit vector and Streaming SIMD (single instruction, multiple data) Extensions, SSE, support. It is not the standard x86_64 instruction set, i.e. Intel/AMD server and Intel Phi processors are not binary compatible.

The Intel Phi coprocessors are also refered to as Intel MIC or their Intel product code names Knights Ferry, KNF, (for the prototype), Knights Corner, KNC, (for the first generation product), and Knights Landing (for the second generation product, to be released).

The Fermilab High-Performance Computing department acquired four servers with four 5110P cards each for investigation and prototyping of high-energy physics, HEP, applications. As part of this activity the Compact Muon Solenoid, CMS, experiment [2] took a closer look at the technology. Other groups investigated the use of the cards for Lattice Quantum Chromo-Dynamics, LQCD, calculations, trigger, detector simulation, accelerator modeling, and other applications.

In this note we report on the CMS investigation and show some of the performance measurments made.

2 Setup

There are four servers with each four Intel 5110P coprocessor cards. Each server has two Intel E5-2620 processor with 6 cores (hyper-threading is turned off) that run at 2.0 GHz. Each server has 32 GBytes of memory, and Infiniband and Ethernet connection (via which two NFS areas are mounted). The servers run Scientific Linux Fermi, SLF, 6.3. The Phi coprocessor cards also run Linux, not SLF but BusyBox with a 2.6 kernel.

One additional server is used to build software (and provides one of the NFS areas). The server has two Intel X5650 processor with 6 cores each (hyper-threading is turned off) that run at 2.67 GHz. It has 12 GBytes of memory. The server has the Intel Manycore Platform Software Stack, MPSS, installed and the Intel Cluster Studio Suite. Former contains the GNU Compiler Collection, GCC, cross compiler for the Intel Phi coprocessor cards. Unfortunately, the GCC cross compiler is based on a rather old version 4.7.0 and has no support for the vector floating point unit of the Phi coprocessor. Cluster Studio contains the Intel cross compilers, icc, icpc, and ifort. The first version of the Intel cross compilers used end of 2013 was version 14.0.0. The last version used in the investigation reported here was version 14.0.2, from March

of 2014.

Access to each server and the Phi coprocessors is scheduled via a batch system. Portable Batch System, PBS, is used to allocate one or more Phi coprocessor cards.

3 CMSSW

The CMS offline software, CMSSW, consists of experiment specific software and third party, "external" packages. End of 2013, CMS had already started building a CMSSW version for the Phi coprocessor cards at CERN. The version of the Intel compiler for the first build was 13.1.3 and the CMSSW version 7_0_0_pre5.

In November 2013 we fetched and installed CMSSW version 7_0_0_pre7 build with the Intel cross compilers version 14.0.0. Most external packages build but only a small number of CMS specific packages. When Intel released version 14.0.2 CMS tried to build the CMSSW version under development at that time, 7_0_0_pre8. More software of the CMS specific packages now compiled but still only 47 of the 2211 shared libraries/plugins in the CMSSW release built.

The build failures investigated further turned out to be due to more complex templates and C++11 extensions. For example, CMS uses the *emplace_hint* function of *std::map* that comes as extension with the C++11 standard but that is not yet available in the Intel compilers used. The Intel compiler also set *__cplusplus* to 1 instead of 201103L in case the *-std=c++11* option is specified. For Phi coprocessor compilation, the Intel compiler is not self-contained but uses include files from GCC 4.7.0 of the MPSS package. GCC 4.7.0 has incomplete, "experimental" C++11 support, particularly for concurrency, one of the main reasons for C++11 use in CMS.

In January 2014 it became clear that the Intel C++ compiler was months away from compiling all (or at least a usefull subset) of a release 7 CMSSW version. Fermilab looked into the Intel port of GCC 4.7.0 for the Phi coprocessor, extracted the patch and started to integrate it into GCC 4.8.2. (GCC 4.8.1 was the compiler used by CMS on x86_64 at the time.) The CMS offline group was already thinking to move to GCC 4.9. Problem reports on the Internet showed that people had started porting GCC 4.9 to the Phi coprocessor. CMS offline management ask to abandon the GCC 4.8.2 porting activity and started on GCC 4.9 at CERN instead.

As of July 2014, using Intel compilers version 15.0.0 beta 2, 45% of the CMS specific packages build [3]. CMS reported the compiler problems to Intel with each new compiler version and provided the compiler group at Intel with a copy of a CMSSW release earlier this year. There is no GCC 4.9 port for the Phi coprocessor available yet.

4 Sherpa

Sherpa [4] is a Monte Carlo generator used in CMS and throughout HEP. It is one of the external packages of CMSSW but is independent of it and can be downloaded, build, and run without CMSSW.

Sherpa is one of the few event generators that can make use of multi-core CPUs (without the user running multiple instances). The software can be compiled with multi-threading

support and/or Message Passing Interface, MPI. MPI is the method recommended by the Sherpa team to use a large number of cores (or nodes) in a very scalable approach. The CPU intensive phase-space integration has been run and its scalability measured on high-performance computing/leadership class computing facilities, utilizing 16,000 cores [5]. The authors of [5] also investigated speeding up Monte Carlo phase-space integration by using Graphics Processing Unit, GPU, and Intel Phi coprocessor cards using a special version of MadGraph [6] and Sherpa.

With neither CMS simulation nor reconstruction program available, using Sherpa in a stand-alone mode is then the next best approach to investigate the Intel Phi coprocessor for CMS. For this Sherpa version 2.0.0 was downloaded and build with MPI support using the Intel compiler suite version 14.0.2.

The Z-boson production example at the Large Hardon Collider, LHC, in next-to-leading order QCD precision using BlackHat was used. The jet multiplicity was, however, reduced from four to three to adjust the execution time to less than a day for a single core modern x86_64 CPU. (This would keep multi-core execution times at a durations where the execution startup overhead could be ignored.)

A Sherpa process for the Z-boson production example uses about 750 MBytes of memory. For the Intel Phi 5110P coprocessor card with 8 GBytes of memory this would allow less than ten running instances or MPI processes. (Part of the 8 GBytes memory is also used by the BusyBox operating system.) This wouldn't leave much to investigate. We thus expanded the evaluation from just the Phi coprocessor to also the Intel compiler suite.

For the investigation we decided to make three builds: Using the default Scientific Linux GCC compilers for x86_64, using the Intel compiler suite for x86_64, and using the Intel compiler suite for the Phi coprocessor. For the GCC x86_64 MPI build MPICH version 3.1 is used while for the Intel compiler builds the MPI provided with the Intel compiler suite is used.

Using BlackHat for loop matrix elements in Sherpa requires the BlackHat software package and QD, a quad-double precision datatype software package. The default GCC version for Scientific Linux 6.4 is 4.4.7. It compiled QD, BlackHat, and Sherpa without error.

The Intel icpc compiler failed to compile one file for x86_64 and another for the Phi coprocessor. For file "qd_real.cpp" the "mcpcom" process seemed to be getting into an infinite loop. The file was compiled with an -O1 flag, instead of the default -O2 optimization. For the Phi coprocessor build file "fpu.cpp" failed to compile with error 113, "incomplete type is not allowed". The file was compiled with the g++ compiler of GCC 4.7.0 from MPSS. Building BlackHat failed on file "eval_param.cpp" with "error #803: member ... explicitly instantiated more than once". The Intel support web site acknowledges the issue and advises to use the -wd803 compiler flag to overcome the error. With the flag BlackHat compiled fine. However, during execution of Sherpa, loading of the BlackHat shared object library failed when instantiating the template class reported in the #803 error. For both x86_64 and the Phi coprocessor QD and BlackHat were thus compiled with GCC[1] and Sherpa with the Intel compiler suite.

The executables are all about 1 MByte (except for the x86_64 GCC MPI one which is about a third the size). Most execution code is dynamically loaded from over 200 shared libraries.

[1]For the Phi coprocessor build GCC 4.7.0 of the MPSS package was used while for the x86_64 build GCC 4.4.7 from Scientific Linux.

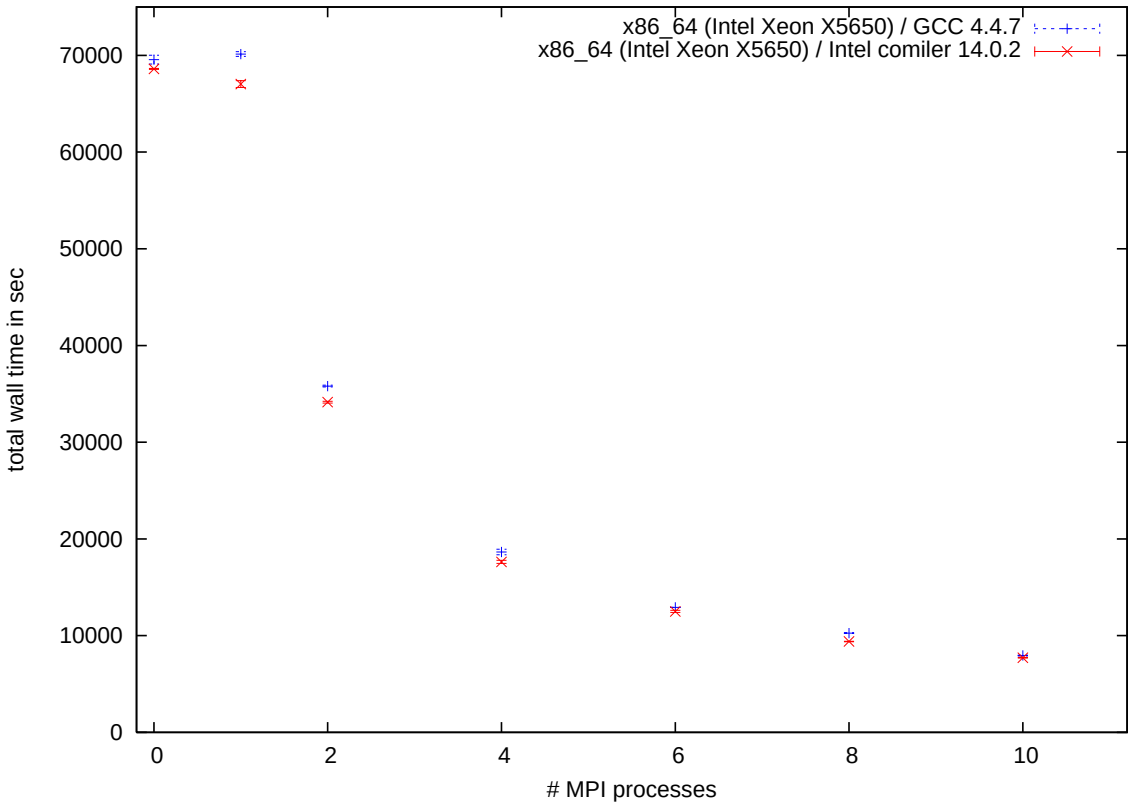

Figure 1: Total wall time to complete the Z-boson production phase-space integration. The points at zero are for the non-MPI executables. Error bars show the standard deviation of the multiple timing measurments made for each number of MPI processes.

A non-MPI executable takes 710 MByte (x86_64 GCC), 740 growing to 4800 MByte (x86_64 Intel), and 710 MByte (Phi Intel) of memory during execution. An MPI executable requires 730 MByte (x86_64 GCC), 780 growing to 4900 MByte (x86_64 Intel), and 710 MByte (Phi Intel) of memory for one process execution. There is a small rise in memory of about 10 MByte per MPI process. The x86_64 Intel build Sherpa runs with about 740 MByte (780 MByte MPI) until about 75 sec before finishing. At that time the process specific/private memory ("anonymous memory" in pmap) starts to grow to over 4 GByte (for MPI in each process). The GCC build Sherpa processes do not increase in memory before finishing. This is most likely an Intel compiler issue. We plan to rebuild and test this with the next compiler release.

While analyzing the Sherpa memory usage in more detail we realized that about 600 MBytes of process memory is used by shared libraries (Sherpa, BlackHat, QD, matrix element to be integrated, etc.), i.e. can be shared by MPI processes. This then allows to run about 50 MPI processes on the Phi coprocessor. (Considering also the growth of 10 MByte with each MPI process).

The x86_64 versions were executed and timed on the build server with the two 6-core Intel Xeon X5650 CPUs. The non-MPI executable took 19 hours 15 minutes 13.035 seconds (GCC

build) and 18 hours 59 minutes 10.077 seconds (Intel build) to complete. This makes the Intel compiler build about 1.5% faster. There was no machine/architecture specific tuning done during compilation, only default -O2 optimization.

The MPI executables show excellent scaling with an overhead of 1.4% (GCC build) and 1.1% (Intel build) per MPI process. Figure 1 shows the measured wall time and Fig. 2 the total CPU time of all the MPI processes as as function of number of MPI processes. The non-MPI execution is plotted at zero, Intel build measurements are x-offset by +0.25, the y-axis in Fig. 2 is zero suppressed.

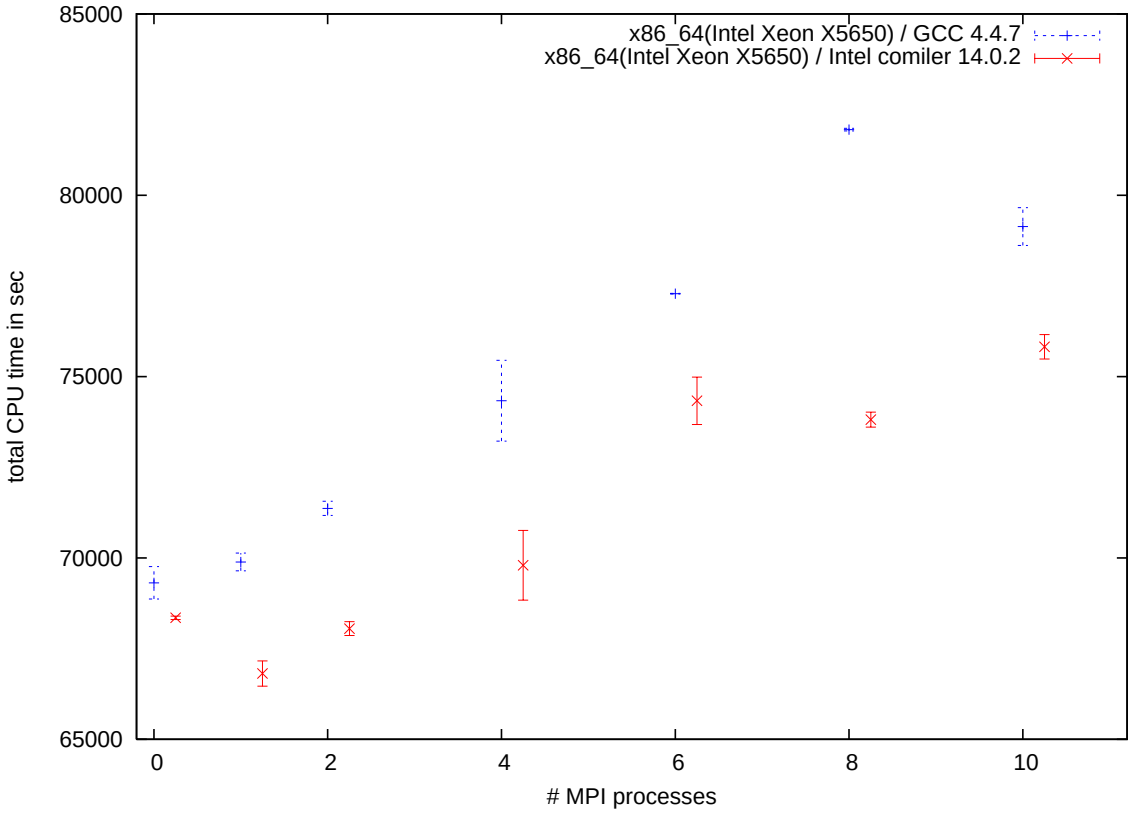

Figure 2: Total CPU time used by all processes. The points at zero are for the non-MPI executables. Error bars show the standard deviation of the multiple timing measurments made for each number of MPI processes.

In case of the x86_64 Intel builds, the 1-process MPI executable run faster than the non-MPI executable by about 2%. All timing was derived from multiple executions with the standard deviation shown by the error bar in the figures. The difference is thus significant, about 4σ and all individual runs of the MPI executable are faster than all the non-MPI runs. We did not find the reason for this. The GCC and Phi coprocessor builds behaved as expected, with the 1-process MPI executable slightly slower than the non-MPI executable.

The Phi coprocessor builds run much slower compared to the x86_64 builds. We expected a significantly slower execution, by as much as an order of magnitude, as the Intel Phi 5110P

coprocessor is based on mid 1990s Pentium technology. However, we did not appreciate how big of an effect out-of-order execution, branch prediction, pre-fetching, etc. has for large modern HEP programs with linked data structures across large amounts of memory. Our expectation was based on the product of CPU inprovements since the mid 1990s. However, this is wrong, since today's programs execute much slower on older CPUs than the speed up of old programs on newer CPUs. The first execution we aborted after a few weeks of running with steady but slow progress. We then used a check point in the program to estimate the total execution time on the Phi coprocessor. For the non-MPI executable this gives us a total CPU time estimate of 48 month! We used check points and their fraction of the total execution time from x86_64 runs for all Phi coprocessor timings.

Figure 3 shows the timing of the Phi coprocessor builds together with the x86_64 builds. The y-axis has a logarithmic scale. Phi coprocessor builds run about $1,750$ times slower than the x86_64 builds. The larger number of Phi coprocessor cores, 60 versus 12, does not compensate for the slower execution. It is clear that using the Phi coprocessor in this way is not efficient.

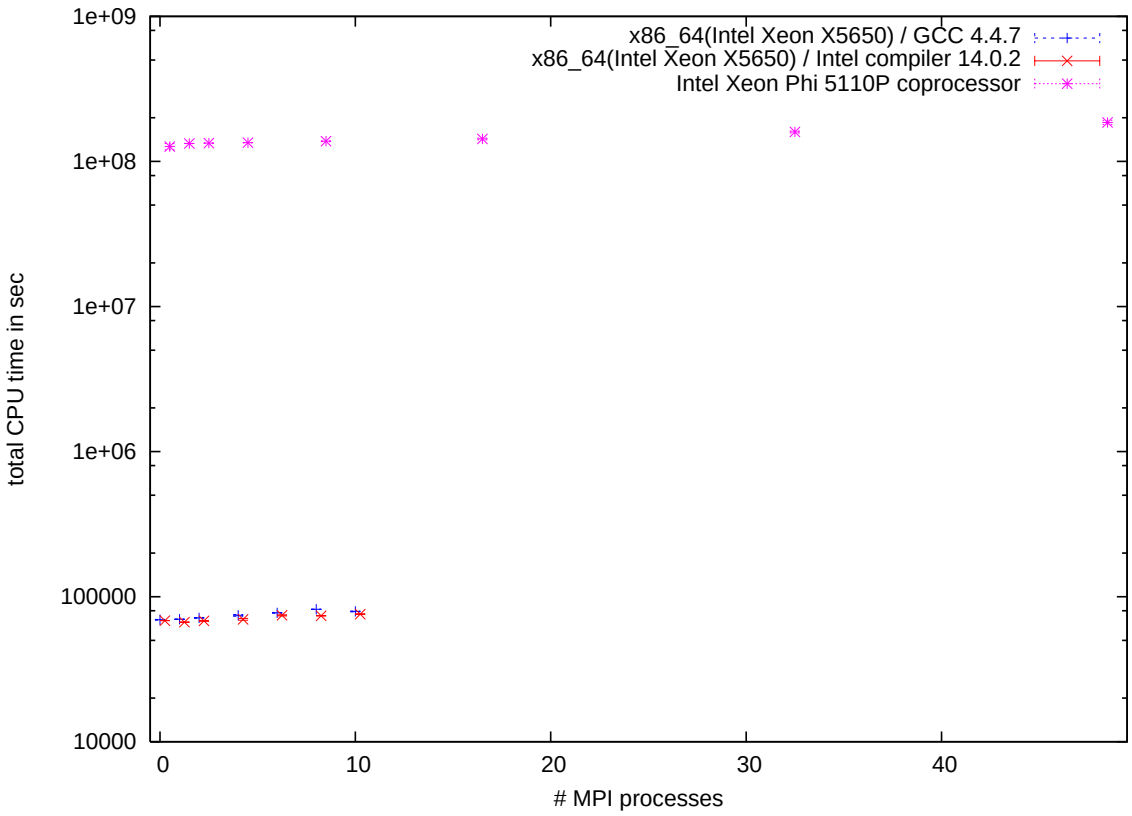

Figure 3: Total CPU time used by all processes. The points at zero are for the non-MPI executables. Error bars show the standard deviation of the multiple timing measurments made for each number of MPI processes.

5 Conclusions

The current generation of Intel Xeon Phi coprocessors have the capability of large floating point performance. They are not x86_64 binary compatible and require an Intel provided compiler. The Intel compiler does not (yet) support the C++11 extensions the CMS software uses. It fails to compile half the CMS software packages as of July 2014.

The compiler has also problems with other high-energy-physics software. Compared to a GCC build x86_64 executable and Intel build x86_64 executable is only negligibly faster, within the speed up expected from architecture-specific tuning or by using a more recent GCC release.

A standard Monte Carlo application was found to execute $1,750$ times slower on the Intel Knights-Corner Xeon Phi coprocessor core than on an Intel Nehalem-based Xeon core. To use Phi coprocessors effectively, high-energy-physics software needs to be modified/adapted to off-load compute-intensive tasks onto the Phi coprocessor such that calculations with localized floating-point data can execute in parallel.

References

[1] http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

[2] The Compact Muon Solenoid, CMS, experiment uses a large general-purpose detectors to analyze high-energy particle collisions of the Large Hadron Collider, LHC, at CERN in Switzerland. http://cms.web.cern.ch/, http://cmsdoc.cern.ch/cms/cpt/tdr/

[3] S. Muzaffar, private communication (2014).

[4] T. Gleisberg *et al.*, J. High Energy Phys. 02 (2009) 007, http://sherpa.hepforge.org/

[5] C. Bauer *et al.*, arXiv:1309.3598 [hep-ph]

[6] J. Alwall *et al.*, arXiv:1405.0301 [hep-ph], http://madgraph.hep.uiuc.edu/